Decentralised Negotiation for Multi-Object Collective Transport with Robot Swarms

Guillermo Legarda Herranz
IRIDIA-CoDE
Université libre de Bruxelles
Brussels, Belgium
guillermo.legarda.herranz@ulb.be

Sabine Hauert
Bristol Robotics Laboratory
University of Bristol
Bristol, United Kingdom
sabine.hauert@bristol.ac.uk

Simon Jones
Bristol Robotics Laboratory
University of Bristol
Bristol, United Kingdom
simon2.jones@bristol.ac.uk

***Abstract*—Recent developments of robot swarms with richer capabilities for sensing and manipulation of the environment have opened the door to more complex applications of swarm robotics. The introduction of such swarms in intralogistics, where workers are still at risk of injury, is of particular interest. We present a method to control a swarm of robots to simultaneously transport multiple items that are too heavy or too large for a single robot to carry. We introduce a decentralised negotiation strategy based on inter-robot communication, which allows the robots to coordinate with subgroups of the swarm. We then use genetic programming to evolve behaviour tree controllers that generate the desired action of each robot, which is then fed to the negotiation strategy to produce the final output.**

***Index Terms*—swarm robotics, collective transport, negotiation**

I. Introduction

According to the latest data from Eurostat, almost 2.7 in every 100 workers in the European Union experienced workplace injuries or illnesses in the transportation and storage sector in the year 2019 [1]. To prevent accidents, autonomous mobile robots may assist in the transport of heavy items, such as pallets and packages [2]. However, these systems can be costly and require additional infrastructure or detailed setup. Additionally, they are usually controlled by a central unit, and are therefore highly sensitive to its failure.

In this paper, we propose a method for collective transport of heavy items using robot swarms. The goal of swarm robotics is to design decentralised systems that are robust, scalable and flexible [3]. By using readily replaceable, cost-effective robots, swarms are an attractive solution to operate in dangerous environments, like natural disaster sites and outer space.

Tuci *et al.* [4] differentiate between pushing-only strategies for collective transport, where robots have no means of holding an object, and grasping strategies, where robots have specialised hardware to grasp or lift objects. The main advantage of grasping strategies is that they prevent damages to the objects by avoiding dragging them, which makes them more suitable for real-world applications. However, they pose the additional challenge of requiring a coordination strategy for the robots to align their forces without releasing the object. This implies that the robots need to position themselves correctly about an object before attaching themselves to it and attempting to move it. As a consequence, it is commonly assumed that the robots are attached to the object from the start. We highlight two notable examples of this approach. Campo *et al.* [5] proposed a negotiation method where robots use LEDs to communicate their estimate of the goal direction. Each robot then uses the directions indicated by the rest of the robots to improve their own estimation. Ferrante *et al.* [6] developed a socially-mediated negotiation mechanism, where robots use direct communication to compute a common direction of motion while taking into account how much of the environment each robot is able to perceive.

Despite the many successful implementations of collective transport strategies that exist in the literature [4], we notice an absence of methods that explicitly consider the simultaneous transport of multiple large items. To the best of our knowledge, the only existing work that has tackled this issue was developed by Hamouda [7]. In their implementation, however, the robots are only able to push an item in a straight line towards the goal, and therefore have no obstacle avoidance capabilities during transportation. Consequently, we present a negotiation strategy for a swarm of robots with grasping capabilities to simultaneously transport large items while safely navigating the environment.

In the context of negotiation, every robot in a group suggests an action based on its own perception of the environment. Considering all the actions suggested, the group reaches a consensus and acts accordingly. However, computing the action that each robot must suggest such that the swarm achieves the desired global behaviour is challenging. In swarm robotics, while the design of the individual robot controllers is often tackled manually, automatic design methods are a convenient alternative, as they offer repeatability of the process and reduced costs [8]. We are interested in off-line automatic design, in which controllers are designed before the swarm is deployed in the environment, without any need for human intervention [9]. In particular, we take the modular approach, where a set of low-level behaviours are combined to form controllers [10].

In automatic design, the selection of the controller architecture determines the set of tools that might be used to discover a solution. We are especially interested in behaviour trees (BTs). Originally designed to specify the behaviour of non-player characters (NPCs) in video-games [11], BTs exhibit

978-1-6654-8217-2/22/$31.00

multiple properties that make them suitable for robot control applications [12]. They are modular, that is, each subtree constitutes its own legal BT, which allows for testing and reusing of parts of a BT separately. They are also human readable, which makes the analysis of the resulting robot behaviour more straightforward than, for example, neural networks. Furthermore, they are reactive, and thus robots may quickly react to changes in the environment.

Behaviour trees have been shown to provide designers with a high degree of control over the granularity of the low-level behaviours and promote understanding of the resulting robot behaviour. In one of the earlier works on the control of real robots using BTs, Scheper *et al.* [13] used an evolutionary algorithm to generate BT controllers for a flapping-wing micro-aerial-vehicle (MAV). Jones *et al.* [14] later introduced the use of genetic programming (GP) techniques to evolve BT controllers for foraging tasks with a swarm of kilobots, as well as a method for online evolution of BTs in the Xpuck Teraflop swarm [15]. Finally, behaviour trees have also been used in automatic modular design methods [16]. In a similar fashion to Jones *et al.* [14], we use GP to evolve the BT controllers that generate the inputs for the negotiation strategy. We develop and evaluate our methodology in simulation[1].

II. Methods

A. Task

We conceive an environment consisting of a 5 m $\times$ 5 m arena with a nest region on the right-hand-side ($+x$) and a number of items scattered about it, which always require more than one robot to transport (see Figure 4). In our collective transport task, the objective is to have the robot swarm move the items towards the nest. When an item is placed at the nest, it is reset to its starting position to allow the swarm to operate indefinitely, and its distance travelled is accumulated. We therefore define the raw fitness of the task for each item, i, as the normalised displacement:

$$f_{raw,i} = \max\left\{\frac{x_{final,i} - x_{start,i} + \Delta x_i}{t_{sim} v_{max}}, 0\right\}, \qquad (1)$$

where $x_{start,i}$ and $x_{final,i}$ are the start and end positions, Δx_i is the net distance travelled by the item towards the $+x$ end of the arena before the last time it was placed at the nest, t_{sim} is the simulation runtime and $v_{max} = 0.2$ ms^{-1} is the maximum speed of the robots. It is worth pointing out that $f_{raw,i}$ is always nonnegative, even if the net displacement of the item is towards the $-x$ end of the arena. The total fitness of the task is then

$$f = \frac{1}{M}\sum_{i=1}^{M} f_{raw,i}, \qquad (2)$$

where M is the number of items in the arena. Consequently, $f \in [0, 1]$, where a fitness of 0 indicates that no items were transported towards the nest, while a fitness of 1 indicates that all M items were constantly moving towards the nest at the maximum velocity possible. We identify two main phenomena that cause the fitness of a run to be lower than the theoretical upper limit of 1. First, robots in the swarm inevitably spend time exploring the environment. Second, interference between robots hinders the motion of those carrying an item towards the nest. Therefore, we expect even high-performing swarms to achieve fitness values lower than 1.

We consider simulated holonomic, circular robots with a diameter of 0.25 m. They are equipped with various sensing and actuating capabilities to interact with the rest of the swarm and the environment. They can detect nearby objects up to a distance of 0.15 m and extract their position and orientation, which allows them to navigate while avoiding collisions. We also simulate point-like idealised fiducial markers that the robots can detect up to 1 m away and thus compute their relative position and orientation, as well as obtain data stored in them. We cover the circumference of each robot with such markers, which allows them to detect and identify each other. All the robots have access to a common direction (east) and can also exchange messages with neighbours that are within a 1 m radius. Finally, we assume that the actions of the robots in our swarm are synchronised. While this concession implies that the robots have access to a global clock, it allows us to greatly simplify both the presentation and the implementation of our coordination strategy, which is the topic that we want to address in this paper.

The nest can be identified by the robots through the same idealised fiducial markers used for neighbour detection. In this case, it is the right wall of the arena that is covered with markers. The robots know that they are at the nest when their distance to any nest marker is under a certain threshold. The items are simulated as regular polygons with a predefined set of *lifting points*, which are once again identifiable by means of our fiducial markers. Upon detection, these markers also return the total number of lifting points of the item. Once all the lifting points of an item are covered by robots, the robots can attach themselves to the item and transport it. In order for the robots to be able to communicate with each other while transporting an item, all lifting points are within a 1 m radius (the communication range of the robots) of the rest of the lifting points. Additionally, to increase the chances of a robot encountering a lifting point, we assume that additional markers cover the remaining surface of the items. Each of these markers holds a vector that points in the direction of the nearest lifting point.

B. Control

A control methodology that implements a coordination strategy is required for the swarm to collectively transport items. The goal of such a coordination strategy is to allow groups of robots carrying an item, which we refer to as *porters*, to safely navigate the environment and move independently of the rest of the swarm.

We therefore conceive a two-stage control loop. In the first stage, given its perception of the environment, each robot

[1]The entire codebase developed for this work is available on GitHub: https://github.com/glegarda/collective-transport.git

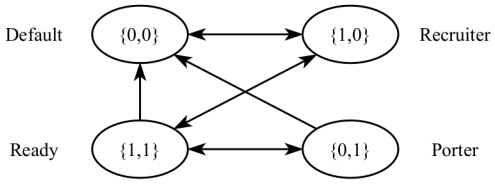

Fig. 1. Available transitions of the $\{L, G\}$ tuple, where L and G are binary variables that denote whether the robot has detected an item and is part of a group, respectively. Each value of the tuple corresponds to a different robot state.

executes a BT to generate its desired outputs. In the second stage, each robot communicates with its neighbours to share its desired outputs and negotiate the action to be executed. The BT is generated using GP, while the negotiation stage remains hardcoded.

The rest of this section is devoted to describing the implementations of the BTs and the negotiation strategy. The negotiation strategy constitutes our main contribution and motivates some aspects of the BT implementation.

1) Negotiation: In every iteration, each robot computes two desired outputs: the velocity in its local frame, $\boldsymbol{v_{vote}}$, and p_{vote}, a discrete variable representing an attachment (1) or detachment (-1) command from an item. Since all robots have access to a common direction, it then rotates $\boldsymbol{v_{vote}}$ to the global frame, $\boldsymbol{v'_{vote}}$, and broadcasts a message, which we call a *vote*. Each vote is defined as a tuple $m = \{i, l, g, r'_{vote}, \angle b'_{vote}, p_{vote}\}$, where i is the robot identifier and l is the item identifier, which is 0 when the robot is not at a lifting point or when it is carrying an item. We denote robots for which $l \neq 0$ as *recruiters*, since they represent robots that are waiting for others to transport an item. The variable g is the group identifier, which is 0 by default. When a robot is at a lifting point, g is set to l if, in one controller iteration, it receives as many messages with a value of l equal to its own as the number of porters required to transport the item, minus one. The values of r'_{vote} and $\angle b'_{vote}$ correspond to the magnitude and orientation of $\boldsymbol{v'_{vote}}$, respectively.

Each robot then receives the votes cast by its neighbours and computes the goal velocity in the global frame, $\boldsymbol{v'_{goal}}$, and the attachment command, p_{goal}. We define these as

$$p_{goal} = p_{vote} + (1 - \delta_{0g}) \cdot \sum_{j=1}^{s} p_{vote,j} \delta_{g_j g}, \tag{3}$$

$$\boldsymbol{v'_{goal}} = \left(\boldsymbol{v'_{vote}} + (1 - \delta_{0g}) \cdot \sum_{j=1}^{s} \boldsymbol{v'_{vote,j}} \delta_{g_j g} \right) \cdot \delta_{0 p_{goal}}, \tag{4}$$

where s is the number of messages received. The Kronecker deltas δ_{0g} and $\delta_{g_j g}$ force every robot to coordinate with other robots in the same, nonzero group, thus allowing them to coordinate with a subgroup of the swarm. The other Kronecker delta, $\delta_{0 p_{vote}}$, prevents any moving robot from attaching itself to an object. We choose to prioritise the attachment command because we believe that a successful controller should rely on the motion of the robot for most purposes.

Since all robots in the same group (i.e., all porters of the same item) compute p_{goal} and $\boldsymbol{v'_{goal}}$ according to (3) and (4), they all obtain the same values, so long as they form a fully-connected network. This is achieved by design of the items, through correct placement of the lifting point markers, and therefore is of no concern to the robots. However, the computed p_{goal} and $\boldsymbol{v'_{goal}}$ do not necessarily coincide with any of the individual votes, which differentiates our negotiation strategy from traditional voting schemes. Each robot in the group can then rotate $\boldsymbol{v'_{goal}}$ back to its local frame to obtain $\boldsymbol{v_{goal}}$. Since the velocity magnitudes generated by the BT are potentially unbound, $|\boldsymbol{v_{goal}}|$ values are first saturated at 1.0 and then mapped to the $[0, v_{max}]$ range. The result of all the robots moving according to $\boldsymbol{v_{goal}}$ is that the group moves at the same speed, in the same global direction. In a real-world deployment, this would prevent potential damages to both the item and the grasping hardware of the robots.

The states and transitions available to each robot in this negotiation framework are summarised in Figure 1. We define two variables: L, which is 1 when l is not 0, and 0 otherwise; and G which, similarly, is 1 when g is not 0, and 0 otherwise. Each possible combination of the $\{L, G\}$ tuple then represents a different state. Given (3) and (4), robots only consider the votes cast by their neighbours when they are in the *ready* or *porter* states, where $\delta_{0g} = 1$.

2) Behaviour trees: Behaviour trees are directed acyclic graphs of nodes and edges. Periodically, the root node sends a *tick* signal to its children, which is propagated down the tree. Once the signal reaches the leaf nodes, these interact with the environment and return one of three possible states: *success*, *failure* or *running*. The result is then propagated back up the tree until it reaches the root node.

BTs interact with the environment through the *blackboard*. The blackboard consists of a list of entries, each of which contains information about a different aspect of the environment as perceived by each robot. Entries can either be scalar values or vectors, where vectors are given in polar form in the heading-oriented frame of the robot. Additionally, every entry has an access level. Leaf nodes may obtain the value of an entry with read level, or may also modify an entry with write level. Table I shows the blackboard constructed for this work, of which each robot holds its own copy. The magnitude of $\boldsymbol{v_{prox}}$ is 0 when no obstacles are detected and 1 when an obstacle is adjacent to the robot. The vectors $\boldsymbol{v_{attr}}$, $\boldsymbol{v_{recr}}$, $\boldsymbol{v_{home}}$ and $\boldsymbol{v_{lift}}$ are set to $(1, \angle 0)$ so long as the robot does not detect any of their corresponding markers of interest. The vote entries, $\boldsymbol{v_{vote}}$ and p_{vote}, are set to 0 at the beginning of each iteration.

We adopt the methodology proposed by Francesca *et al.* [10] to automatically design control software. We define a set of behaviours and conditions using the blackboard entries available. The GP algorithm then draws from this set to construct BTs during the optimisation process, as described in Section II-C. We define the following set of behaviours:

- Exploration: perform a random walk. If the robot encounters an obstacle, it sets $\boldsymbol{v_{vote}}$ to a unit vector with a

TABLE I
BLACKBOARD ENTRIES.

Name	Access	Description
$\boldsymbol{v_{prox}}$	R	Location of nearby obstacles, if any
$\boldsymbol{v_{attr}}$	R	Location of nearby neighbouring robots, if any
$\boldsymbol{v_{recr}}$	R	Location of recruiting neighbours, if any
$\boldsymbol{v_{home}}$	R	Location of the nearest nest marker, if any
$\boldsymbol{v_{lift}}$	R	Location of the nearest lifting point, if any
s_n	R	Number of neighbouring robots
s_r	R	Number of neighbouring recruiters
s_p	R	Number of porters of the same item
p_{vote}	RW	Voted platform command
$\boldsymbol{v_{vote}}$	RW	Voted velocity
s_{zero}	R	Constant zero scalar
$\boldsymbol{v_{zero}}$	R	Constant zero vector
s_{scr}	RW	Scalar scratchpad
$\boldsymbol{v_{scr}}$	RW	Vector scratchpad

random orientation away from $\boldsymbol{v_{prox}}$.

- Stop: stand still.
- Attraction/Repulsion: move towards or away from neighbours. The robot sets $\boldsymbol{v_{vote}} = \alpha \boldsymbol{v_{attr}} - k\boldsymbol{v_{prox}}$, where $\alpha \in [-5, 5]$ and $k = 5$.
- Recruitment/Anti-recruitment: move towards or away from recruiters. The robot sets $\boldsymbol{v_{vote}} = \alpha \boldsymbol{v_{recr}} - k\boldsymbol{v_{prox}}$, where α and k are defined as above.
- Pick: attach to an item. If the robot is at a lifting point, $p_{vote} = 1$; otherwise, it sets $\boldsymbol{v_{vote}} = \boldsymbol{v_{lift}} - k\boldsymbol{v_{prox}}$, where k is defined as above.
- Place: detach from an item. If the robot is at the nest, $p_{vote} = -1$; otherwise, it sets $\boldsymbol{v_{vote}} = \boldsymbol{v_{home}} - k\boldsymbol{v_{prox}}$, where k is defined as above.

All behaviours return *success* by default. However, if a behaviour sets either $\boldsymbol{v_{vote}}$ or p_{vote}, any subsequent behaviours that attempt to overwrite them in the same iteration will return *running*. We also define the following conditions:

- Neighbour-count: return *success* with probability $z(s_n) = \frac{1}{1+e^{k(l-s_n)}}$, where k and l are 5.3 fixed-point values.
- Recruiter-count: return *success* with probability $z(s_r) = \frac{1}{1+e^{k(l-s_r)}}$, where k and l are 5.3 fixed-point values.
- Porter: if the robot is a porter, return *success* with probability β.
- Nest: if the robot is at the nest, return *success* with probability β.
- Item: if the robot has located an item, return *success* with probability β.
- Fixed-probability: return *success* with probability β.

C. Evolution

We use the openGA library[2] for our implementation of GP. We define each chromosome as a BT controller generated using the syntax of the BehaviorTree.CPP library[3].

The algorithm proceeds as follows. Initially, a population of $N = 30$ individuals is generated using Koza's ramped-half-and-half method with minimum and maximum depths of $d_{min} = 0$ and $d_{max} = 4$, respectively [17]. To create each BT, all the constituent behaviours and conditions defined in Section II-B2, as well as the control-flow nodes introduced by Marzinotto *et al.* [18], including the *Node** extensions, are available. We also define four decorators: *inverter*, which returns *success* when its child returns *failure* and vice-versa; *success*, which always returns *success*; *failure*, which always returns *failure* and *repeat*, which returns *running* until its child node returns *success n* times, or *failure* if its child returns *failure*. In order to limit the search space of the algorithm, the maximum number of children the *selector*, *sequence*, *selector** and *sequence** nodes are allowed to have is limited to four. If any of the selected nodes have parameters, their values are drawn from a uniform probability distribution in the appropriate range.

Once the population is initialised, the algorithm performs genetic operations and selects the individuals for the subsequent generation for $n_{gen} = 400$ generations. For the genetic operations, $N \cdot f_{crossover}$ new individuals are generated by crossover, where $f_{crossover} = 0.7$. Each of the newly generated individuals then undergo parameter, point and subtree mutation with probabilities $p_{param} = 0.05$, $p_{point} = 0.05$ and $p_{subtree} = 0.1$, respectively. From the resulting pool, $n_{elite} = 3$ individuals are transferred to the next generation using elitism. The remaining $N - n_{elite}$ individuals of the new generation are selected using rank selection.

This implementation of GP often suffers from bloat, where the size of the BTs grows uncontrollably, but their fitness does not reflect a similar growth [17]. To limit its effects and keep the resulting BTs readable, we implement covariant parsimony pressure, which dynamically sets the parsimony coefficient during a run to penalise larger trees [19].

To evaluate the fitness of each individual, we built a lightweight 2D simulator of our environment using the Box2D physics engine[4]. For every fitness evaluation, we initialise an instance of the environment with an arena, three two-porter items and 16 robots. The items are placed on the left-hand side of the arena, while the robots are randomly scattered around the right-hand side of the arena (see Figure 4). Each controller is run for 120 s. The simulator runs one iteration of the complete control loop every 0.1 s, computing a goal velocity $\boldsymbol{v_{goal}}$. We then apply white noise with a standard deviation of 5% to every velocity [20]. The physics is updated with a frequency of 30 Hz, that is, three times per control iteration. The fitness of the controller is then obtained as the mean fitness, as per (2), of five independent runs.

To assess our methodology, we conduct 10 independent evolutionary runs. In each run, for each generation, we record the highest-performing controller, its raw fitness and size, and the mean raw fitness and size of the population. The resulting BTs may contain superfluous substructures, that is, sections of the BT that can never be ticked and therefore do not affect the outcome. To further improve readability of the trees for

[2]https://github.com/Arash-codedev/openGA.git

[3]https://github.com/BehaviorTree/BehaviorTree.CPP.git

[4]https://github.com/erincatto/box2d.git

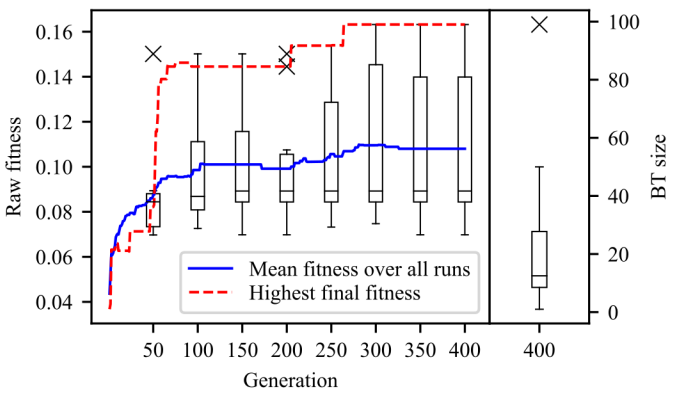

Fig. 2. (Left) evolution of the raw fitness of the highest-performing controllers of the 10 evolutionary runs. The blue line shows the mean raw fitness, the red line shows the raw fitness of the controller with highest final raw fitness and box and whisker plots are shown every 50 generations. (Right) size of the highest-performing BTs in the last generation of all 10 runs.

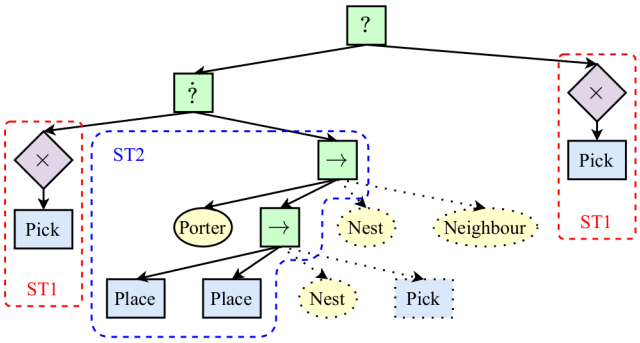

Fig. 3. Fittest solution of the evolutionary runs. The control-flow nodes used are *sequence* (→), *selector* (?) and *selector** (?̇) nodes. Lilac rhombuses are decorators; in this case, *failure*. Dotted lines represent nodes removed by pruning. ST1 and ST2 are subtrees of interest (see text).

analysis, we mechanically prune the BTs by following the automatic procedure outlined in Jones *et al.* [15] to remove said substructures. We verify that the resulting BT is equivalent to the original by running five instances of the simulator with the robots in the same initial conditions and equal seeds for the pseudorandom number generators employed. We log the positions of every robot in the swarm at every controller timestep and verify that the resulting files are identical.

III. Results

Figure 2 shows the evolution of the raw fitness of the highest-performing controllers across the 10 independent runs. The lines indicate the run that obtained the controller with the highest final raw fitness and the mean raw fitness over all runs. We also show box and whisker plots every 50 generations. Additionally, the figure shows a box and whisker plot representing the sizes of the highest-performing BTs in the last generation of the 10 runs, before we apply the pruning technique. The results shows that the swarm is able to evolve a solution to collectively transport items towards the nest.

The low medians in Figure 2 suggest that the evolutionary runs were dominated by low-performance controllers. By inspecting the solutions obtained from each run, we concluded that half of the controllers were executing the pick behaviour exclusively. Indeed, by executing the pick behaviour, the robots move from obstacle to obstacle and, if they ever run into a lifting point, they wait for other robots to fill the rest of the lifting points (in the case considered, there is only another lifting point). Once all the robots required to transport the item are located under the corresponding lifting points, they attach themselves to the item and continue moving between obstacles. As a result, the swarm is rewarded for transporting the items, even though the robots do not actively move towards the nest.

The spread of the box and whisker plots reflect how the remaining controllers were able to achieve the task with various levels of success. In order to understand what the swarm is actually doing in the more successful cases, we take advantage of the readability of behaviour trees. We consider the highest-performing controller obtained by the evolutionary algorithm, which obtained a raw fitness value of 0.163.

As shown in Figure 3, if we consider constituent behaviours and conditions as single nodes of the BT, the tree has 15 nodes. To prune the BT, we only need to make use of the fact that, if one node in a *sequence* always returns *running*, no additional nodes in that *sequence* are ever ticked. Since, as mentioned in Section II-B2, no behaviour may overwrite $\boldsymbol{v_{vote}}$ or p_{vote}, the second of the two consecutive place behaviours always returns *running*. We therefore remove all remaining nodes in the *sequence*. By applying the same principle to the parent *sequence*, we obtain a pruned BT with 11 nodes.

Analysing the resulting behaviour of the swarm then becomes straightforward. We recognise two subtrees of interest. In the first subtree (ST1), a robot executes the pick behaviour to explore the environment and find the items. Thus, it moves forward while performing obstacle avoidance and, when an item is found, it moves towards the nearest lifting point and waits for enough robots to attach itself to the item. In the second subtree (ST2), a robot checks if it is a porter. If it is, it executes the place behaviour; that is, it heads towards the nest and, once reached, it detaches from the item. If it is not a porter, the BT ticks the second instance of ST1. Since ST2 contains two place behaviours in a *sequence*, ST2 always returns *running* if the robot is a porter, or *failure* if the robot is not a porter. Consequently, since ST1 and ST2 are the children of a *selector**, the BT skips the first ST1 and the robot executes the place behaviour as long as it is a porter.

Figure 4 shows a timelapse of a run of the swarm in which the robots execute the controller shown in Figure 3. Besides the behaviour deduced from the analysis of the BT, we can observe how the swarm is able to carry two items simultaneously towards the nest. Additionally, the figure shows how the robots indeed need time to explore the environment and locate the items, and also how porters can run into other robots, which constitute obstacles in their way to the nest. We therefore argue that, while seemingly low when

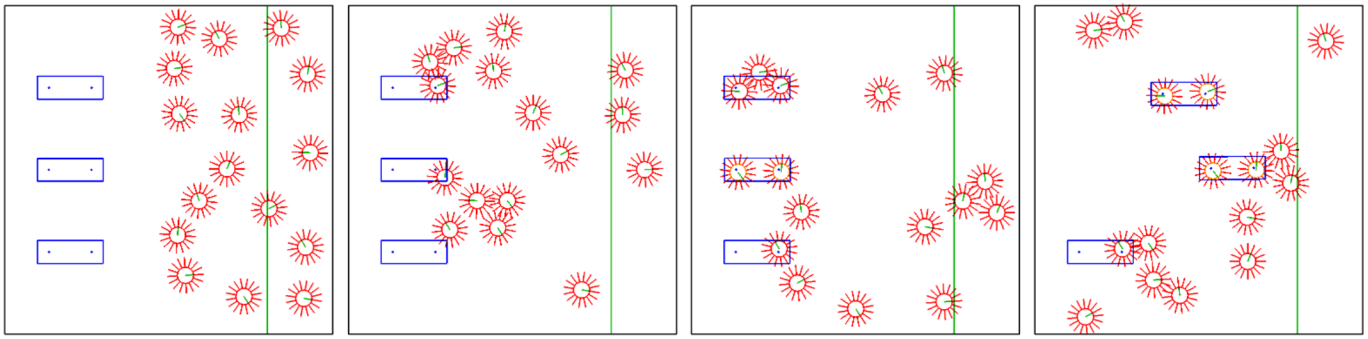

Fig. 4. Timelapse of the behaviour of the swarm using the controller shown in Figure 3. From left to right, the swarm is initialised, the lifting points are located, enough porters are gathered under the lifting points and two items are simultaneously transported towards the nest.

compared with the theoretical maximum fitness of 1, a fitness of 0.163 is in line with our expectations for a high-performing swarm. Nevertheless, further experimentation with different population sizes, number of items and sizes of the items would be needed to study their effect on the performance of the swarm. This is left as a possibility for future work.

IV. Conclusions

We have presented a decentralised negotiation strategy based on direct communication that allows the robots in a swarm to collectively transport multiple large items simultaneously in a safe manner. We introduce it as the second stage of the controller where, in the first stage, behaviour trees are used to generate the desired outputs. We have also shown how these BTs may be successfully evolved using genetic programming techniques. The results show that, after pruning, the evolved BTs are human readable, which makes the deduction of the behaviour of the robots straightforward.

Acknowledgments

The project has received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (Demiurge: grant agreement No 681872) and from Belgium's Wallonia-Brussels Federation through a ARC Advanced Project 2020 (Guaranteed by Optimization).

References

[1] Eurostat. (2022, Mar.) Accidents at work - statistics by economic activity. https://ec.europa.eu/eurostat/statistics-explained/index.php?title=Accidents_at_work_-_statistics_by_economic_activity.

[2] K. Azadeh, R. de Koster, and D. Roy, "Robotized warehouse systems: Developments and research opportunities," *ERIM Report Series Research in Management Erasmus Research Institute of Management*, no. ERS-2017-009-LIS, 2017.

[3] E. Şahin, "Swarm robotics: from sources of inspiration to domains of application," in *SAB 2004*, ser. LNCS, E. Şahin and W. M. Spears, Eds., vol. 3342. Berlin, Germany: Springer, 2005, pp. 10–20.

[4] E. Tuci, M. H. M. Alkilabi, and O. Akanyeti, "Cooperative object transport in multi-robot systems: A review of the state-of-the-art," *Front. Robot. AI*, vol. 5, p. 59, 2018.

[5] A. Campo, S. Nouyan, M. Birattari, R. Groß, and M. Dorigo, "Negotiation of goal direction for cooperative transport," in *ANTS 2006*, ser. LNCS, M. Dorigo, L. M. Gambardella, M. Birattari, A. Martinoli, R. Poli, and T. Stützle, Eds., vol. 4150. Berlin, Germany: Springer, 2006, pp. 191–202.

[6] E. Ferrante, M. Brambilla, M. Birattari, and M. Dorigo, "Socially-mediated negotiation for obstacle avoidance in collective transport," in *DARS 10*, ser. STAR, A. Martinoli, F. Mondada, N. Correll, G. Mermoud, M. Egerstedt, M. A. Hsieh, L. E. Parker, and K. Støy, Eds., vol. 83. Berlin, Germany: Springer, 2013, pp. 571–583.

[7] A. I. Hamouda, "Cooperative transport in swarm robotics. multi object transportation," Master's thesis, AUC, Cairo, Egypt, 2018.

[8] S. Hauert, S. Mitri, L. Keller, and D. Floreano, "Evolving cooperation: From biology to engineering," in *The Horizons of Evolutionary Robotics*, ser. Intelligent Robotics and Autonomous Agents, P. Vargas, E. Di Paolo, I. Harvey, and P. Husbands, Eds. Cambridge, MA, USA: MIT Press, 2014, pp. 203–217.

[9] M. Birattari, A. Ligot, D. Bozhinoski, M. Brambilla, G. Francesca, L. Garattoni, D. Garzón Ramos, K. Hasselmann, M. Kegeleirs, J. Kuckling, F. Pagnozzi, A. Roli, M. Salman, and T. Stützle, "Automatic off-line design of robot swarms: a manifesto," *Front. Robot. AI*, vol. 6, p. 59, 2019.

[10] G. Francesca, M. Brambilla, A. Brutschy, V. Trianni, and M. Birattari, "AutoMoDe: a novel approach to the automatic design of control software for robot swarms," *Swarm Intell.*, vol. 8, no. 2, pp. 89–112, 2014.

[11] D. Isla, "Handling complexity in the Halo 2 AI," in *GDC 2005*, vol. 12. London, United Kingdom: Game Developers Conference (GDC), 2005.

[12] M. Colledanchise and P. Ögren, *Behavior Trees in Robotics and AI: An Introduction*, 1st ed., ser. Chapman & Hall/CRC Artificial Intelligence and Robotics Series, R. Yampolskiy, Ed. Boca Raton, FL, USA: CRC Press, 2018.

[13] K. Y. W. Scheper, S. Tijmons, C. C. de Visser, and G. C. H. E. de Croon, "Behavior trees for evolutionary robotics," *Artif. Life*, vol. 22, no. 1, pp. 23–48, 2016.

[14] S. Jones, M. Studley, S. Hauert, and A. Winfield, "Evolving behaviour trees for swarm robotics," in *DARS 13*, ser. SPAR, R. Groß, A. Kolling, S. Berman, E. Frazzoli, A. Martinoli, F. Matsuno, and M. Gauci, Eds., vol. 6. Cham, Switzerland: Springer, 2018, pp. 487–501.

[15] S. Jones, A. Winfield, S. Hauert, and M. Studley, "Onboard evolution of understandable swarm behaviors," *Adv. Intell. Syst.*, vol. 1, no. 6, p. 1900031, 2019.

[16] J. Kuckling, V. van Pelt, and M. Birattari, "AutoMoDe-Cedrata: automatic design of behavior trees for controlling a swarm of robots with communication capabilities," *SN Comput. Sci.*, vol. 3, p. 136, 2022.

[17] J. R. Koza, *Genetic programming: on the programming of computers by means of natural selection*, 1st ed. Cambridge, MA, USA: MIT Press, 1992, a Bradford Book.

[18] A. Marzinotto, M. Colledanchise, C. Smith, and P. Ögren, "Towards a unified behavior trees framework for robot control," in *ICRA 2014*. Piscataway, NJ, USA: IEEE, 2014, pp. 5420–5427.

[19] R. Poli and N. F. McPhee, "Covariant parsimony pressure for genetic programming," University of Essex, Essex, United Kingdom, Tech. Rep. CES-480, 2008.

[20] N. Jakobi, P. Husbands, and I. Harvey, "Noise and the reality gap: the use of simulation in evolutionary robotics," in *ECAL '95*, ser. LNAI, F. Morán, A. Moreno, J. J. Merelo, and P. Chacón, Eds., vol. 929. Berlin, Germany: Springer, 1995, pp. 704–720.